



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/420,798	10/19/1999	YOSHIHIKO IMAMURA	SON-1661	3308

7590 03/18/2005

RONALD P KANANEN ESQ  
RADER FISHMAN & GRAUER  
THE LION BUILDING  
1233 20TH STREET NW SUITE 501  
WASHINGTON, DC 20036

EXAMINER
----------

OPIE, GEORGE L

ART UNIT	PAPER NUMBER
----------	--------------

2126

DATE MAILED: 03/18/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

<b>Office Action Summary</b>	<b>Application No.</b>	<b>Applicant(s)</b>	
	09/420,798	Imamura	
	<b>Examiner</b>	<b>Art Unit</b>	
	George L. Opie	2126	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

#### Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136 (a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).

#### Status

- 1) ☒ Responsive to communication(s) filed on 31 January 2005.
- 2a) ☐ This action is **FINAL**.                      2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

#### Disposition of Claims

- 4) ☒ Claim(s) 23-27, 29-36, 38-39 and 41-44 is/are pending in the application.
- 4a) Of the above claim(s) ☐ is/are withdrawn from consideration.
- 5) ☐ Claim(s) ☐ is/are allowed.
- 6) ☒ Claim(s) 23-27, 29-36, 38-39 and 41-44 is/are rejected.
- 7) ☐ Claim(s) ☐ is/are objected to.
- 8) ☐ Claim(s) ☐ are subject to restriction and/or election requirement.

#### Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☐ The drawing(s) filed on ☐ is/are objected to by the Examiner.
- 11) ☐ The proposed drawing correction filed on ☐ is: a) ☐ approved b) ☐ disapproved.
- 12) ☐ The oath or declaration is objected to by the Examiner.

#### Priority under 35 U.S.C. § 119

- 13) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d).
- a) ☐ All b) ☐ Some \* c) ☐ None of the CERTIFIED copies of the priority documents have been:
1. ☐ received.
  2. ☐ received in Application No. (Series Code / Serial Number) ☐.
  3. ☐ received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

\* See the attached detailed Office action for a list of the certified copies not received.

- 14) ☐ Acknowledgement is made of a claim for domestic priority under 35 U.S.C. & 119(e).

#### Attachment(s)

- 14) ☒ Notice of References Cited (PTO-892)                      17) ☐ Interview Summary (PTO-413) Paper No(s) ☐.
- 15) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)                      18) ☐ Notice of Informal Patent Application (PTO-152)
- 16) ☐ Information Disclosure Statement(s) (PTO-1449) Paper No(s) ☐.
- 19) ☒ Other: Text doc for USP5,787,272 included in mailing

Art Unit: 2126

## DETAILED ACTION

This Office Action is responsive to the Amendment, filed 31 January 2005, in which claims 23-24, 26, 36, 38-39 and 43-44 were amended.

**1. Request for copy of Applicant's response on floppy disk:**

Please help expedite the prosecution of this application by including, along with your amendment response in paper form, an electronic file copy in WordPerfect, Microsoft Word, or in ASCII text format on a 3½ inch IBM format floppy disk.

Please include all pending claims along with your responsive remarks. Only the paper copy will be entered -- your floppy disk file will be considered a duplicate copy. Signatures are not required on the disk copy. The floppy disk copy is not mandatory; however, it will help expedite the processing of your application. Your cooperation is appreciated.

2. The U.S. Patents used in the art rejections below have been provided as text documents which correspond to the U.S. Patents. The relevant portions of the text documents are cited according to page and line numbers in the art rejections below. For the convenience of Applicant, the cited sections are highlighted in the *text documents*.

3. Claim Rejections - 35 U.S.C. § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

4. Claims 23-27, 29-36, 38-39 and 41-44 are rejected under 35 USC §103(a) as being unpatentable over the Admitted Prior Art (APA) provided in the Application background in view of **Gupta et al. (U.S. Patent 5,787,272)**.

As to claim 23, the APA teaches a "multiprocessor which is comprised of a plurality of CPUs connected via a common bus and executes a plurality of mutually independent programs in parallel", Application-page3 wherein

Art Unit: 2126

a first processor element "processor element 111", page4 of said plurality of processor elements for executing a first user program "instruction codes ... prg A are successively executed", page5 of a plurality of user programs, said first processor element executes a wait instruction, said wait instruction suspends processing of said first user program "when an instruction code 'wait (Prg D)' is executed in the processor element 111, the processing ... enters a synchronization waiting state", page7 and

a second processor element "processor element 114", p7 of said plurality of processor elements for executing a second user program "Prg D" of said plurality of user programs, said second processor element executes a wait release instruction "code 'end' of the subprogram Prg D" said wait release instruction commands said first processor element to resume said processing of said first user program "message indicating the completion of the subprogram Prg D is notified to the processor element 111 . . . As a result, the processor element 111 releases the synchronization waiting state and executes the next instruction code."

Although the APA does not explicitly disclose the first processor executing a "program end instruction" to cause resumption of the second program, it would have been obvious for one skilled in the art, from the APA's "instruction code 'end'" teaching wherein processor 114 conveys a release/resume command to processor 111 when processor 114 has executed an "end" instruction, to provide that the first processor would likewise employ this function to have the second program resume its execution.

The APA does not explicitly disclose the second processor element continuing processing of the second program after executing the wait release instruction.

Gupta teaches a system in which "the processor will be able to continue executing instructions", p3 35-41 which corresponds to the processor element continuing processing of its program after executing the wait release instruction. It would have been obvious to combine Gupta's teachings with the APA because the procedure for the processors to carry on execution will optimize the computing resources as "the processors will have to spend very little or no time waiting for each other", p5 1-9, and thereby achieving greater parallel processing efficacy.

As to claim 24, the APA teaches a first processor element "processor element 111", page4 of said plurality of processor elements for executing a first user program "instruction codes ... prg A are successively executed", page5 of a plurality of user programs, said first processor element executes a wait instruction, said wait instruction suspends processing of said first user program

Art Unit: 2126

"when an instruction code 'wait (Prg D)' is executed in the processor element 111, the processing ... enters a synchronization waiting state", page 7

a second processor element "processor element 114", p7 of said plurality of processor elements for executing a second user program "Prg D" of said plurality of user programs, said second processor element executes a wait release instruction "code 'end' of the subprogram Prg D" said wait release instruction commands said first processor element to resume said processing of said first user program "message indicating the completion of the subprogram Prg D is notified to the processor element 111 . . . As a result, the processor element 111 releases the synchronization waiting state and executes the next instruction code."

The APA does not explicitly disclose the second processor element continuing processing of the second program after executing the wait release instruction.

Gupta teaches a system in which "the processor will be able to continue executing instructions", p3 35-41 which corresponds to the processor element continuing processing of its program after executing the wait release instruction. It would have been obvious to combine Gupta's teachings with the APA because the procedure for the processors to carry on execution will optimize the computing resources as "the processors will have to spend very little or no time waiting for each other", p5 1-9, and thereby achieving greater parallel processing efficacy.

As to claim 25, the APA teaches a "multiprocessor which is comprised of a plurality of CPUs" using VLSI *on a single chip for parallel processing*.

As to claim 26, the APA (page 6) teaches the processor executing instructions without suspending program execution after signaling a "release" instruction.

As to claim 27, the APA (page 3) teaches parallel processing programs with "communication between processes" by sending messages over a common bus.

As to claim 29, see the background details on the instruction code "end" executed in the processor elements 111 through 114 on page 7 of the APA.

As to claims 30-31, the APA teaches a first storage means "common memory 15", p4 and second storage means "local memory 32" that correspond to the processing means "processor elements 111 to 114", p5 reading programs from the first storage means "user programs read from the common memory ... and successively supplies instruction codes of the user program stored in the local memory 32 to the processor core 31 for execution.", page 4.

Art Unit: 2126

As to claim 32, the APA teaches the system "reads the user programs stored in the common memory 15 into the local memories 32", page 5, until the "end" instruction terminates the process.

As to claim 33, the APA (pages 5-7) teaches the "wait release" instruction executed by the corresponding processor element as claimed.

As to claim 34, the APA teaches "instruction code 'gen(Prg\_B)' is executed in the processor element 111, . . . Then the subprogram Prb B stored in the common memory 15 is read into the local memory 32 of the processor element 112" pp5-6.

As to claim 35, the APA (page 5) teaches an "arbiter 16" that corresponds to the program execution assigning means and its claimed functions.

As to claims 36, 38-39 and 41-42, note the rejections of claims 23-24, 26, 33 and 35 respectively. Claims 36, 38-39 and 41-42 are the same as claims 23-24, 26, 33 and 35, except claims 36, 38-39 and 41-42 are method claims and claims 23-24, 26, 33 and 35 are apparatus claims.

As to claim 43, see the discussion of claim 23 supra. Claim 43 is functionally equivalent to claim 23, but for the limitation that the other processing means enters a waiting state when executing the release instruction, which would have been an obvious variation from the claim 23 recitations. Having the other processor pause for synchronization when releasing the "wait" of the first process would naturally have flowed from the APA's parallel process coordination teachings.

As to claim 44, note the rejection of claim 26 above. Claim 44 is the same as claim 26, except claim 44 is a computer program product claim and claim 26 is a method claim.

5. The prior art of record and not relied upon is considered pertinent to the applicant's disclosure. Specifically, the below reference(s) will also have relevancy to one or more elements of the Applicant's claimed invention as follows:

U.S. Patent No. 6,330,619 to Kruezburg which teaches the parallel processing that advances programs while issuing codes to coordinate the processes;

U.S. Patent No. 6,263,406 to Uwano et al. which teaches the parallel execution of programs progressing and activating synchronization controls;

U.S. Patent No. 5,781,775 to Ueno which teaches the task execution management for continuing jobs while coordinating the requisite parallelism;

Art Unit: 2126

U.S. Patent No. **5,742,824** to Kosaka which teaches the process maintaining its running after supplying/signaling concurrency controls; and,  
U.S. Patent No. 5,481,747 to Kametami which teaches the parallel processing system with functionality for processors to send coordination interactions while carrying on processing.

**6. Response to Applicant's Arguments:**

Applicant's remarks, filed 31 January 2005, have been considered but they are moot in view of the new grounds of rejection.

**7. Contact Information:**

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system.

Status information for published applications may be obtained from either Private-PAIR or Public-PAIR.

Status information for unpublished applications is available through Private-PAIR only.

For more information about the PAIR system, see <http://pair-direct.uspto.gov>.

Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

All responses sent by U.S. Mail should be mailed to:

**Commissioner for Patents  
PO Box 1450  
Alexandria, VA 22313-1450**

Hand carried responses should be delivered to the *Customer Service Window* (Randolph Building, 401 Dulany Street, Alexandria, Virginia 22314) and, if submitting an electronic copy on floppy or CD, to expedite its processing, please notify the below identified examiner prior to delivery, so that the Applicant can "handoff" the electronic copy directly to the examiner.


Art Unit: 2126

The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

All OFFICIAL faxes will be handled and entered by the docketing personnel. The date of entry will correspond to the actual FAX reception date unless that date is a Saturday, Sunday, or a Federal Holiday within the District of Columbia, in which case the official date of receipt will be the next business day. The application file will be promptly forwarded to the Examiner unless the application file must be sent to another area of the Office, e.g., Finance Division for fee charging, etc.

Any inquiry of a general nature or relating to the status of this application should be directed to the Group receptionist at **(571) 272-2100**.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to George Opie at (571) 272-3766 or via e-mail at *George.Opie@uspto.gov*. Internet e-mail should not be used where sensitive data will be exchanged or where there exists a possibility that sensitive data could be identified unless there is an express waiver of the confidentiality requirements under 35 U.S.C. 122 by the Applicant. Sensitive data includes confidential information related to patent applications.

  
MENG-LI T. AN  
SUPERVISORY PATENT EXAMINER  
TECHNOLOGY CENTER 2100



**MAIL WITH OFFICE ACTION**

**ATTACHMENT  
FOR PTO-326**

**U.S. PATENT  
5,787,272**

TITLE: Method and apparatus for improving synchronization time  
in a parallel processing system  
INVENTOR(S): Gupta, Rajiv, Ossining, NY, United States  
Epstein, Michael Abraham, Spring Valley, NY, United  
States

PATENT ASSIGNEE(S): Philips Electronics North America Corporation, New  
York, NY, United States (U.S. corporation)

	NUMBER	KIND	DATE	
PATENT INFORMATION:	US 5787272		19980728	<--
APPLICATION INFO.:	US 1997-871562		19970610	(8)
RELATED APPLN. INFO.:	Continuation of Ser. No. US 1995-488311, filed on 7 Jun 1995, now abandoned which is a division of Ser. No. US 1994-189269, filed on 31 Jan 1994, now abandoned which is a continuation of Ser. No. US 1991-689383, filed on 22 Apr 1991, now abandoned which is a division of Ser. No. US 1988-227276, filed on 2 Aug 1988, now abandoned			
DOCUMENT TYPE:	Utility			
FILE SEGMENT:	Granted			

	NUMBER	DATE	CLASS	INVENTOR
REFERENCED PATENT:	US 5151991	Sep 1992	395/706.000	Iwasawa et al.
	US 5222229	Jun 1993	395/553.000	Fukuda et al.
	US 5317734	May 1994	395/706.000	Gupta
	US 5434995	Jul 1995	395/553.000	Oberlin et al.

PRIMARY EXAMINER: Kim, Kenneth S.

LEGAL REPRESENTATIVE: Barschall, Anne E.

NUMBER OF CLAIMS: 11

EXEMPLARY CLAIM: 1

NUMBER OF DRAWINGS: 6 Drawing Figure(s); 6 Drawing Page(s)

#### ABSTRACT:

A barrier is used to synchronize parallel processors. The barrier is "fuzzy", i.e. it includes several instructions in each instruction stream. None of the processors performing related tasks can execute an instruction after its respective fuzzy barrier until the others have finished the instruction immediately preceding their respective fuzzy barriers. Processors therefore spend less time waiting for each other. A state machine is used to keep track of synchronization states during the synchronization process.

This is a continuation of prior application Ser. No. 08/488,311, filed on Jun. 7, 1995, now abandoned, which is a divisional application of application Ser. No. 08/189,269, filed on Jan. 31, 1994, now abandoned, which is a continuation of application Ser. No. 07/689,383, filed Apr. 22, 1991, now abandoned, which is a divisional of application Ser. No. 07/227,276, filed Aug. 2, 1998, now abandoned.

#### BACKGROUND OF THE INVENTION

##### 1. Field of the Invention

The invention relates to synchronizing parallel processors. In particular the invention relates to the use of barriers for such synchronization.

##### 2. Prior Art

Known parallel processing systems execute computer code which has been converted into parallel instruction streams. Dividing computer code into parallel instruction streams has been described, for instance, in M. Wolfe et al. "Data Dependence and Its Application to Parallel Processing", International Journal of Parallel Programming, Vol. 16, No. 2 April 1987 pp. 137-178, and H. Stone, High Performance Computer Architecture, (Addison Wesley 1987) pp. 321, and 336-338. Some of the streams have lexically forward dependences and/or loop carried dependences. The concept of lexically forward dependences is described

in R. Cytron, "Doacross: Beyond Vectorization for Multiprocessors", 1986 IEEE International Conference on Parallel Processing, pp. 836-844, especially at page 838. Loop carried dependences are described in M. Wolfe et al. The lexically forward and loop carried dependences lead to a requirement for synchronization between the instruction streams.

Using "barriers" allows for such synchronization. Barriers are points in the respective parallel instruction streams where the respective parallel processors have to wait to synchronize with each other. The use of barriers for synchronization is described in P. Tang et al., "Processor Self-Scheduling for Multiple-Nested Parallel Loops", Proc. 1986 Int. Conf. Parallel Processing, Aug. 1986, pp. 528-535.

A detailed description of a parallel processing system which uses such stopping points for synchronization can be found in U.S. Pat. Nos. 4,344,134; 4,365,292; and 4,412,303 all issued to Barnes, or Barnes et al.

In the known parallel processing systems, the individual processors must spend time waiting for each other while they are attempting to synchronize. This makes the systems inefficient.

#### SUMMARY OF THE INVENTION

It is an object of the present invention to make parallel processing systems more efficient by reducing the amount of time that individual processors must spend waiting for each other.

This object is achieved by a synchronization apparatus which synchronizes parallel processors so that at least one of the processors executes at least one non-idling instruction while awaiting synchronization with at least one other processor.

This object is further achieved by identifying certain regions of code in the respective instruction streams. The regions are referred to herein as "shaded" and "unshaded". The shaded regions are defined herein as "fuzzy" barriers. A processor begins to attempt synchronization upon reaching a respective shaded region. Synchronization is achieved when no processor executes an instruction following its respective shaded region until all processors performing related tasks have finished all instructions in the unshaded region preceding their respective corresponding shaded region.

The object is still further achieved by an apparatus which coordinates synchronization information between parallel processors and which uses a state machine to keep track of synchronization information.

Further objects and advantages will be apparent from the remainder of the specification and from the claims.

#### BRIEF DESCRIPTION OF THE DRAWING

FIG. 1a is a flow chart which describes a method for compiling source code to identify shaded and unshaded regions.

FIG. 1b is a flow chart describing steps for reordering code.

FIG. 2 is a system diagram showing a parallel processing system.

FIG. 3 is a block diagram of circuit for synchronizing parallel processors.

FIG. 4 is a state diagram.

FIG. 5 is a detailed diagram of the contents of box 304.

#### DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

FIG. 1a is a flow chart showing compilation of source code to create shaded regions.

In box 101, the compilation starts with source code. An example of some C source code, suitable for parallel processing follows:

---

```
int a[10][4];  
.  
.  
for (j=2; j<10; j++)  
  for (i=2, i<5; i++)  
    a[j][i] = a[j-1][i+1] +i*j,
```

In box 102, the compilation identifies parts of the code which can be executed on separate processors. Box 102 uses the method as described in the above-mentioned book by H. Stone, and article by M. Wolfe et al. In the source code example, the inner loop can be executed in parallel on separate processors. The code for the inner loop would then look as follows:

P1(i=2)	P2(i=3)	P3(i=4)
<pre> . . . (for (j=2, j&lt;10; j++)   { a[j][2]=a[j-1][3]+2*j; barrier; } </pre>	<pre> . . . for (j=2, j&lt;10, j++)   { a[j][3]=a[j-1][4]+3*j; barrier; } </pre>	<pre> . . . (for (j=2, j&lt;10, j++)   { a[j][4]=a[j-1][5]+4*j; barrier; } </pre>

The barriers were inserted because of loop carried dependences. In other words, in the example, the value of `a[2][3]` computed by processor P2 in the first iteration of the loop is needed by processor P1 in the second iteration. In the prior art, each of the three processors would have to wait in each loop until each of the other processors reached the point marked barrier.

In box 103, the compilation generates intermediate code, using standard techniques. In what follows, the intermediate code will be expressed in a standard notation called "three address code". This code and techniques for generating it are described A. Aho et al, Principles of Compiler Design, (Addison Wesley 1979) Ch. 7.

In the example, the intermediate code for the three processors will be the same except for the value of "i" which is initialized to 2, 3, and 4 for processors P1, P2, and P3, respectively.

**Box 104 identifies shaded and unshaded regions. The shaded regions will constitute fuzzy barriers. In other words, as in the case of the traditional barrier, when a processor reaches a shaded region it will want to synchronize. However, in contrast with the prior art, in the case of the fuzzy barrier, or shaded region, the processor will be able to continue executing instructions while waiting to synchronize. The unshaded regions will constitute areas where the processors do not seek to synchronize.**

After box 104, the intermediate code will be:

Comment: Let A be the base address of the array a

```

j = 2
L1:  T1 = j - 1
      T2 = 16 * T1
      T3 = T2 + A
      T4 = (i+1) * 4
      T5 = i * j

```

Comment: unshaded region

```

I1:  T6 = T4[T3] + T5
      /*T6=a[j-1][i+1]+i*j
      */
      T7 = 16 * j
      T8 = T7 + A
      T9 = i * 4

```

```

I2:      T9[T8] = T6      /* a[j][i] = T6
                               */
        j = j + 1
        if j<10 to to L1

```

Box 104 finds these shaded and unshaded region as follows.

The default is for instructions to be in a shaded region. This default is set because the processor can never stall while executing instructions in the shaded region. Shaded regions are therefore preferred.

Finding the unshaded part includes two main steps. First, the first and last instructions with lexically forward dependences and/or loop carried dependences are identified as unshaded. Then all of the instructions between those first and last instructions are also unshaded. In the example, I1 and I2 are the only instructions with loop carried dependences. During the execution of instruction I1, the processor accesses a value that was computed by some other processor in a previous iteration. During execution of instruction I2, a value that will be used by some other processor in a subsequent iteration is stored in the array. Therefore I1, I2, and all of the instructions between them are unshaded.

In executing the code, the parallel processors will be "synchronized" if no processor executes an instruction in the unshaded region following a shaded region, until all other processors have finished all instructions in the unshaded region preceding the corresponding shaded region. This requirement means that those instructions which result in lexically forward and loop carried dependences cannot be executed until the dependences are resolved.

In box 105, the intermediate code is reordered to achieve greater efficiency. Standard reordering techniques can be used for this purpose. Greater efficiency is achieved as the unshaded regions become smaller, because a processor can never be stalled while it is executing instructions in a shaded region. Therefore the reordering techniques are applied to reduce the number of instructions in the unshaded regions. Therefore, after the reordering, the intermediate code is converted to the following:

Comment: Let A be the base address of the array a

```

        j = 2
L1:      T1 = j - 1
        T2 = 4 * T1
        T3 = T2 + A
        T4 = (i+1) * 4
        T5 = i * j
        T7 = 4 * j
        T8 = T7 + A
        T9 = i * 4
Comment: unshaded region
I1:      T6 = T4[T3] + T5
                               /*T6=a[j-1][i+1]+i*j
                               */
I2:      T9[T8] = T6      /* a[j][i] = T6
                               */
        j = j + 1
        if j<10 to to L1

```

In this reordering, the three instructions between I1 and I2 were moved out of the unshaded region. In this example, the three instructions were moved above I1. In some cases, the same effect may be achieved by moving instructions past the last instruction with a lexically forward or loop carried dependence. In other words, the instructions can be moved out of the unshaded region by moving them upward (above I1) or downward (below I2)

In reading the above intermediate code, the reader should note that the code is part of a loop. Thus, the shaded region after the unshaded region joins the shaded region before the unshaded region, in a subsequent iteration. For example, at the end of the first iteration of the loop the first processor can return to the beginning of the loop and keep executing code. If all of the other processors have finished their respective instructions I2 in their first iterations, the first processor can begin its instruction I1 on its second iteration. Since most instructions are in the shaded region, the processors have to spend very little or no time waiting for each other. In box 106, the intermediate code is assembled. For the above example, the VAX assembly code for each processor is given below. Assembly is a standard process performed by standard compilers. During assembly, instructions can be marked as part of the shaded region by turning on a bit reserved for that purpose in the instruction. This bit will be called the "I-bit" in what follows.

P1(i=2)	P2(i=3)	P3(i=4)
movab		
-172(sp),sp		
	movab	
	-172(sp),sp	
		movab
		-172(sp),sp
movl		
\$2,-4(fp)		
	movl	
	\$2,-8(fp)	
		movl
		\$2,-12(fp)
L21:		
moval		
-172(fp),r0		
	L21:	
	moval	
	-172(fp),r0	
		L21:
		moval
		-172(fp),r0
subl3		
\$1,-4(fp),r1		
	subl3	
	\$1,-8(fp),r1	
		subl3
		\$1,-12(fp),r1
ashl		
\$4,r1,r1	ashl	
	\$4,r1,r1	ashl
		\$4,r1,r1
addl2		
r1,r0	addl2	
	r1,r0	addl2
		r1,r0
ashl		
\$1,-4(fp),r1		
	mull3	
	\$3,-8(fp),r1	
		ashl

```

                                $2,-12(fp),r1
addl3
    r1,12(r0),r0
        addl3
            r1,16(r0),r0
                addl3
                    r1,20(r0),r0
moveal
    -172(fp),r1
        moveal
            -172(fp),r1
                moveal
                    -172(fp),r1
ashl
    $4,-4(fp),r2
        ashl
            $4,-8(fp),r2
                ashl
                    $4,-12(fp),r2
addl2
    r2,r1    addl2
                r2,r1    addl2
                    r2,r1
movl
    r0,8(r1) movl
                r0,12(r1)
                    movl
                        r0,16(r1)
incl
    -4(fp)    incl
                -8(fp)    incl
                    -12(fp)
cmpl
    -4(fp),$10
        cmpl
            -8(fp),$10
                cmpl
                    -12(fp),$10
jlss
    L21      jlss
                L21      jlss
                    L21

```

One reordering technique, which can be used in box 105 is described in the flowchart of FIG. 1b. FIG. 1b uses the notation J.sub.LFD to refer to instructions not involved in lexically forward or loop carried dependences and J.sub.LFD to refer to instructions involved in lexically forward or loop carried dependences. All instructions of the type J.sub.LFD are candidates for moving out of the unshaded region. In general, given two instructions, J.sub.i and J.sub.i+1, in that order, then J.sub.i+1 can be moved above J.sub.i, if the following conditions are true:

Condition 1: J.sub.i does not read from a memory location that J.sub.i+1 writes to; and

Condition 2: J.sub.i does not write to a memory location that J.sub.i+1 reads from.

FIG. 1a also assumes an unshaded region having a sequence of instructions J.sub.1, J.sub.2, J.sub.3, . . . , J.sub.N.

Box 150 assigns to J.sub.i the first instruction of the type J.sub.LFD. Box 151 assigns to J.sub.j the first instruction in the unshaded region preceding J.sub.i. Box 152 loops through instructions J.sub.j through J.sub.i, testing Condition 1 and Condition 2, for each instruction. If both Condition 1 and Condition 2 are true, for a given instruction, the method takes branch 153. If either or both of Condition 1 and Condition 2 are false, then the method takes branch 154.

Branch 153 leads to box 155, which tests whether J.sub.j is the last instruction in the unshaded region preceding J.sub.i. If the result of the test of box 155 is false, the method takes branch 156 to box 157, where J.sub.j is assigned the next instruction in the unshaded region preceding instruction J.sub.i. After box 157 the method returns to box 152.

If the result of the test of box 155 is true, the method takes branch 158 to box 159. In box 159, instruction J.sub.i is moved out of the unshaded region. The procedure described shows how instructions may be moved up. After box 159, the method moves to branch 154.

If the result of the tests of box 152 are both false, the method takes branch 154 to box 160. In box 160, the method tests whether I.sub.i is the last instruction of the type J.sub.LFD in the unshaded region. If the result of the test of box 160 is true, the method of FIG. 1b is finished 161. If the result of the test of box 160 is false, then the method takes branch 162 to box 163. Box 163 assigns to J.sub.i the next instruction of the type J.sub.LFD. After box 163, the method of FIG. 1b returns to box 151.

By performing the above steps on the example, it is determined that the only two instructions which must be in the unshaded regions are those which are marked I1 and I2.

A procedure similar to that shown in FIG. 1b can be applied to move the remaining instructions, which do not result in lexically forward or loop carried dependences, down and out of the unshaded region. The similar procedure would differ from that described in FIG. 1b only in that, instead of comparing an instruction with all preceding instructions in the unshaded region, the compiler should compare it with all succeeding instructions.

FIG. 2 is a block diagram of a parallel processing system including four parallel processors 201, 202, 203, and 204, with respective instruction memories 205, 206, 207, and 208. There may be an arbitrary number, n, of processors, where n is an integer greater than 2. Four processors are chosen here for ease of illustration. The parallel processors 201, 202, 203, and 204 share a data memory 209. Each processor has a respective barrier unit 210, 211, 212, and 213. Each barrier unit 210, 211, 212, and 213 has four inputs and two outputs. The three inputs from the other processors indicate whether a respective other processor wants to synchronize. These inputs will be referred to herein as WANT.sub.-- IN. The output which goes to the other processors indicates that the respective processor wants to synchronize. These outputs will be referred to herein as WANT.sub.-- OUT. Each barrier unit 210, 211, 212, and 213 also has a respective I input from and a respective STALL output to its respective execution unit 213, 214, 215, and 216.

FIG. 3 shows more detail in one of the parallel processors 201, 202, 203, and 204, including one of the barrier units 210, 211, 212, and 213. The barrier unit is for receiving, processing, and sending synchronization information. The instruction register 301 is shown within an execution unit 328 and is large enough to contain the longest instruction in the relevant instruction set, plus the I-bit 302. In other words, the processor is assumed to be a RISC processor, which executes one instruction per machine cycle. The I-bit is turned on when the instruction in the instruction register 301 is in a shaded region. The I-bit is off when the instruction is in an unshaded region.

Alternatively, the instruction register 301 can be smaller and instructions can take up several words, if logic, not shown, is included for locking out the



I-bit 302 except in the first word of the instruction. Another alternative approach would be to dedicate an entire instruction in each instruction stream for marking the beginnings of the shaded and unshaded regions. Such an approach would require some minor changes to the state machine. This approach would add instructions to the respective instruction streams, but would require fewer changes to existing hardware and machine instruction sets than the I-bit approach.

The mask register 303, is an internally addressed special register, and has  $n-1$  bits, where  $n$  is the number of processors in the system. In the present example, it is assumed that  $n=4$ . Each of the processors contains the apparatus of FIG. 3. Mask register 303 therefore must have 3 bits, to keep track of the other processors in the system. The mask register 303 is used to ignore other processors which are not performing related tasks. A bit of the mask register 303 is turned off when the corresponding other processor is performing a related task. A bit of the mask register 303 is turned on when the corresponding other processor is not performing a related task. Mask register 303 receives its mask bits from a 3-bit input 320. In the example, only three processors are needed to execute the code. Therefore two bits of the mask register 303 will be off at each processor that is running one of the loops. The third bit will be on, so that the processors running the loops ignore the one processor that is not running the loops. The compiler knows which processors are synchronizing at the barrier and thus can generate an instruction which causes appropriate bits to be written to the mask register 303.

The processors which are ignored, as a result of the bits of the mask register 303 being on in one processor, can in turn perform independent tasks, ignoring the one processor by setting their own mask registers. Such independent tasks can include independent synchronization on an independent job requiring parallel processing.

WANT.sub.-- IN is an  $n-1$  bit input for receiving "WANT" bits from the other processors. The WANT bits will be on when the corresponding processors want to synchronize.

Match circuit 304 contains logic for coordinating the bits in the mask register 303 and the WANT bits on input WANT.sub.-- IN. The output of match circuit 304 is called "MATCH" and is on only when all of the relevant other processors want to synchronize.

State machine 305 uses the I-bit and the output MATCH of the match circuit 304 to determine synchronization states. State machine 305 outputs two bits: STALL and WANT.sub.-- OUT. STALL is off when the processor is executing instructions. STALL is turned on to stop the processor from executing instructions.

WANT.sub.-- OUT is turned on when the respective processor wants to synchronize, and is otherwise off.

FIG. 4 is a state diagram for the state machine 305. In this embodiment, the state machine 305 is a Mealy machine. In other words the outputs STALL and WANT.sub.-- OUT can change without the machine changing states. In FIG. 4, inputs to the state machine 305 are indicated in a smaller font and outputs from the state machine 305 are indicated in bold-face italic font.

Each of the processors 201, 202, 203, and 204 includes one of the state machines as described in FIG. 4. In order for these state machines to work, there must be a common clock or alternate means of synchronizing signals between the state machines. For simplicity, the circuitry for synchronizing the state machines 305 is not illustrated in the figures. Transition 401 corresponds to remaining in state 0. The machine stays in state 0, while the I-bit is off. In other words, the processor is executing an unshaded region of code and is not going to a shaded region of code. STALL and WANT.sub.-- OUT are both off.

Transition 402 takes the machine from state 0 to state 1. The machine makes transition 402 when its respective processor is ready to synchronize, but at least one of the other relevant processors is not ready to synchronize, i.e. when the I-bit is on and MATCH is off. The conditions  $I=0$  and  $MATCH=0$  are denoted  $I^*$  and  $MATCH^*$ , respectively, in FIG. 4. During transition 402, WANT.sub.-- OUT is on and STALL is off. In FIG. 4, when STALL or WANT.sub.-- OUT is off, it is simply omitted. Transition 404 keeps the machine in state 1. The machine makes transition 404 so long as it wants to synchronize and has not yet done so, but is still executing instructions. In other words the machine stays in state 1 while the I-bit is on and MATCH is off. During state 1, WANT.sub.-- OUT is on and STALL is off.

Transition 403 takes the machine from state 0 to state 2. The machine makes transition 403 when its respective processor is ready to synchronize, and it is the last of the relevant processors to get to that point. Several processors can reach state 2 simultaneously and are thus several simultaneous "last" processors. State 2 is a state in which the processor is synchronized. When the state machine 305 is making the transition 403, it keeps WANT.sub.-- OUT on. However, it turns WANT.sub.-- OUT off when it reaches state 2. STALL stays off during transition 403 and state 2.

Transition 405 takes the machine from state 1 to state 2. The machine makes transition 405 when the respective processor is still in its shaded region, wanting to synchronize, and all of the other processors have reached their respective shaded regions, i.e. when both the I-bit and MATCH are on. When the machine makes transition 405, it keeps the WANT.sub.-- OUT bit on. STALL is off during transition 405. The WANT.sub.-- OUT bit returns to off, when the machine reaches state 2.

Transition 406 takes the machine from state 1 to state 3. The machine makes transition 406 when it is ready to leave its shaded region, but has not been able to make it to state 2. In other words, the I-bit turns off and MATCH is off. At this point, the respective processor must stall. Therefore both WANT.sub.-- OUT and STALL are turned on.

Transition 407 takes the machine from state 1 to state 0. The machine makes this transition, when MATCH turns on and the relevant processor leaves the shaded region simultaneously. The machine keeps WANT.sub.-- OUT on during transition 407, and turns it off again when it reaches state 0. STALL remains off during transition 407.

Transition 408 takes the state machine 305 from state 2 to state 0. Transition 408 occurs after synchronization, when the I-bit turns off, i.e. when the respective processor leaves a shaded region. During transition 408, WANT.sub.-- OUT and STALL are both off.

Transition 409 keeps the machine in state 2. Transition 409 occurs after synchronization so long as the I-bit remains 1, i.e. so long as the respective parallel processor remains in the shaded region after synchronization. During transition 409, WANT.sub.-- OUT and STALL are both off.

Transition 411 keeps the machine in state 3, i.e. stalled and waiting to synchronize. The machine makes transition 411 so long as MATCH is off. While in state 3 the machine continues to keep both WANT.sub.-- OUT and STALL on.

Transition 410 takes the machine from state 3 to state 0. The machine makes transition 410 when it has succeeded in synchronizing with the other machines and can leave its shaded region, in other words when MATCH turns on. During transition 410, WANT.sub.-- OUT stays on. WANT.sub.-- OUT turns off, once the machine reaches state 0. During transition 410, STALL is off.

FIG. 5 shows the details of box 304. The three bits 501, 502, and 503 of mask register 303 are also shown in FIG. 5. The mask register 303 has three bits because there are three other parallel processors in the system. The three bits of WANT.sub.-- IN are shown as three separate lines WANT.sub.-- IN0, WANT.sub.-- IN1, and WANT.sub.-- IN2. Mask register bit 503 and WANT.sub.-- IN0

are fed to OR gate 504. Mask register bit 502 and WANT.sub.-- IN1 are fed to OR gate 505. Mask register bit 501 and WANT.sub.-- IN2 are fed to OR gate 506. The outputs of OR gates 504, 505, and 506 are fed to AND gate 507. The output of gate 507 is MATCH.

The output MATCH is thus on when all of the other processors, which are not being ignored, want to synchronize. MATCH is thus also on when all of the other processors are being ignored.

What is claimed is:

1. A parallel processor system comprising: first and second data processors for executing first and second sequences of instructions, respectively, in parallel, wherein: the first sequence comprises a concatenation of a first unshaded region of consecutive instructions followed by a second shaded region of consecutive instructions followed by a third unshaded region of consecutive instructions; the second sequence comprises a concatenation of a fourth unshaded region of consecutive instructions followed by a fifth shaded region of consecutive instructions followed by a sixth unshaded region of consecutive instructions; each instruction in the second shaded region is executable independently of any instruction in the fourth and any successive regions in the second instruction stream; each instruction in the fifth shaded region is executable independently of any instruction in the first and any successive regions in the first instruction stream; at least one instruction in the first unshaded region and at least one instruction in the sixth unshaded region have a dependence relationship, or at least one instruction in the third unshaded region and at least one instruction in the fourth unshaded region have a dependence relationship; synchronization means coupled to the processors for controlling synchronization between the processors during execution of said first and second streams of instructions in parallel, so that the first processor exits the second shaded region only after the second processor has completed execution of the instructions in the fourth unshaded region, or so that the second processor exits the fifth shaded region only after the first processor has completed execution of the instructions in the first unshaded region.

2. The system of claim 1, wherein: the synchronization means detects whether a specific first one of the instructions to be executed next by the first processor belongs to the first unshaded region or the third unshaded region; the synchronization means detects whether a specific second one of the instructions to be executed next by the second processor belongs to the fourth unshaded region or the sixth unshaded region; a necessary first condition for the synchronization means to stall the first processor involves that the specific first instruction belongs to the third unshaded region and the specific second instruction belongs to the fourth unshaded region; a necessary second condition for the synchronization means to stall the second processor involves that the specific second instruction belongs to the sixth unshaded region and the specific first instruction belongs to the first unshaded region.

3. The system of claim 2, wherein the synchronization means detects the associated ones of the regions for the first and second specific instructions under control of the first and second sequences of instructions comprising a plurality of respective marking instructions that mark beginnings of respective ones of the shaded and unshaded regions.

4. The system of claim 3, wherein: the synchronization means is operative to create first and second marking bits for the first and second specific instructions, respectively, under control of the marking instructions; each respective one of the first and second marking bits has a first value if the associated region is a shaded one of said regions, and a second value if the associated region is an unshaded one of said regions; the synchronization means comprises: a first barrier unit for control of the first processor and having a first input, a second input, a first output and a second output; a second

barrier unit for control of the second processor and having a third input, a fourth input, a third output and a fourth output; wherein: the first input is coupled to the third output; the second input receives the first marking bit; the third input is coupled to the first output; the fourth input receives the second marking bit; the first output supplies a first WANT.sub.-- OUT signal; the second output is connected to a stall input of the first processor and supplies a first STALL signal to control stalling of the first processor; the third output supplies a second WANT.sub.-- OUT signal; the fourth output is connected to a stall input of the second processor and supplies a second STALL signal to control stalling of the second processor; the first WANT.sub.-- OUT signal and the first STALL signal are logic functions of the first marking bit and the second WANT-OUT signal; and the second WANT.sub.-- OUT signal and the second STALL signal are logic functions of the second marking bit and the first WANT.sub.-- OUT signal.

5. The system of claim 4, wherein each respective one of the first and second barrier units comprises a respective Mealy state machine.

6. The system of claim 2, wherein: the synchronization means detects the associated ones of the regions for the first and second specific instructions under control of first and second marking bit included in the first and second specific instructions, respectively; and each respective one of the first and second marking bits has a first value if the associated region is a shaded one of said regions, and a second value if the associated region is an unshaded one of said regions.

7. The system of claim 6, wherein the synchronization means comprises: a first barrier unit for control of the first processor and having a first input, a second input, a first output and a second output; a second barrier unit for control of the second processor and having a third input, a fourth input, a third output and a fourth output; wherein: the first input is coupled to the third output; the second input receives the first marking bit; the third input is coupled to the first output; the fourth input receives the second marking bit; the first output supplies a first WANT.sub.-- OUT signal; the second output is connected to a stall input of the first processor and supplies a first STALL signal to control stalling of the first processor; the third output supplies a second WANT.sub.-- OUT signal; the fourth output is connected to a stall input of the second processor and supplies a second STALL signal to control stalling of the second processor; the first WANT.sub.-- OUT signal and the first STALL signal are logic functions of the first marking bit and the second WANT-OUT signal; and the second WANT.sub.-- OUT signal and the second STALL signal are logic functions of the second marking bit and the first WANT.sub.-- OUT signal.

8. The system of claim 7, wherein each respective one of the first and second barrier units comprises a respective Mealy state machine.

9. The system of claim 2, wherein: the synchronization means has mask register means to selectively overrule at least the first condition.

10. The system of claim 4, wherein: the synchronization means has mask register means to selectively overrule at least the first condition; and the mask register means comprises: a programmable register for storing a programmable bit; a logic gate with: a first gate input for receiving the second WANT.sub.-- OUT signal; a second gate input for receiving the programmable bit; a gate output connected to the second input for providing a logic combination of the second WANT.sub.-- OUT signal and the programmable bit.

11. The system of claim 7, wherein: the synchronization means has mask register means to selectively overrule at least the first condition; the mask register means comprises: a programmable register for storing a programmable bit; a logic gate with: a first gate input for receiving the second WANT.sub.-- OUT signal; a second gate input for receiving the programmable bit; a gate output

connected to the second input for providing a logic combination of the second  
WANT.sub.-- OUT signal and the programmable bit.

ISSUE U.S. PATENT CLASSIF.:

MAIN: 395/553.000

SECONDARY: 395/560.000; 395/200.780

CURRENT U.S. PATENT CLASSIF.:

MAIN: 713/375.000

SECONDARY: 709/248.000; 713/601.000

INT. PATENT CLASSIF.: [6]

MAIN: G06F015-16

FIELD OF SEARCH: 395/553; 395/560; 395/200.78

ART UNIT: 274

=>